# APPENDIX A.III

Source code file named libcom_tcp.pl.

```
%*******************************************************************************
%   File     : libcom_tcp.pl
%   Primary Authors  : Adam Cheyer, David Martin
%   Purpose : TCP instantiation of lowlevel communication primitives for OAA
%   Updated : 01/98
%
%   ---------------------------------------------------------------------
%   Unpublished-rights reserved under the copyright laws of the United States.
%
%
%   Unpublished Copyright (c) 1993-98, SRI International.
%   "Open Agent Architecture" and "OAA" are Trademarks of SRI International.
%   ---------------------------------------------------------------------
%
%



%*******************************************************************************
%* RCS Header and internal version
%*******************************************************************************

:- module(com,
        [com_Connect/2,
         com_Disconnect/1,
         com_ListenAt/2,
         com_SendData/2,
         com_SelectEvent/2,
         com_AddInfo/2,
         com_GetInfo/2]).

% rcs version number
rcsid(libcom_tcp, '$Header:
/tmp_mnt/home/zuma1/martin/OAA/agents/beta/prolog/RCS/com_tcp.pl,v 1.10
1998/05/06 22:35:36 martin Exp $').

:- use_module(library(sets)).
:- use_module(library(tcp)).
:- use_module(library(basics)).
:- use_module(library(lists)).
:- use_module(library(charsio)).  % for sprintf and with_output_to_chars
:- use_module(library(ask)).      % for ask_oneof
:- use_module(library(environ)).  % read environment vars
:- use_module(library(files)).        % can_open_file
:- use_module(library(strings)).  % for concat

:- dynamic
      com_connection_info/5,  % id, commtype, client/server, commInfo, status
        com_already_loaded/1. % filename



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_Connect(+ConnectionId, ?Address)
% purpose: Given a connection ID and an address, initiates a client connection
% remarks:
```

```
%  - if Address is a variable, instantiates the Address by using
%     com_ResolveVariables, which looks in a setup file, command line, and
%     environment variables for the required info.
%  - stores the connection info for connection ID in com_connection_info/5.
%  - fails if connection can't be made
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_Connect(ConnectionId, tcp(Host,Port)) :-
      ground(ConnectionId),
      % if variable address, look it up...
      ((var(Host) ; var(Port)) ->
         com_ResolveVariables([
            [cmd('-oaa_host',Host), cmd('-oaa_port', Port)],
            [env('OAA_HOST', Host), env_int('OAA_PORT', Port)],
            [setup('setup.pl', oaa_host, Host),
             setup('setup.pl',oaa_port, Port)]
          ])
      | true),

      tcp_connect(address(Port, Host), RootConnection),
      assert(com_connection_info(ConnectionId, tcp, client,
            [addr(tcp(Host,Port)),
             oaa_host(Host),oaa_port(Port),connection(RootConnection)],
            connected)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_Disconnect(+ConnectionId)
% purpose: Given a connection ID of type 'client', shuts down the connection.
% remarks: Succeeds silently if there is not an open connection having the
%          given id.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_Disconnect(ConnectionId) :-
      ground(ConnectionId),
      com_connection_info(ConnectionId, tcp, client, _Info, connected),
      com_GetInfo(ConnectionId, connection(Connection)),
      tcp_shutdown(Connection),
      retract(com_connection_info(ConnectionId,tcp,client,_Info,connected)),
      !.
com_Disconnect(_ConnectionId).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_ListenAt(+ConnectionId, ?Address)
% purpose: Given a connection ID and an address, initiate a server connection
% remarks:
%  - if Address is a variable, instantiates the Address by using
%     com_ResolveVariables, which looks in a setup file, command line, and
%     environment variables for the required info.
%  - stores the connection info for connection ID in com_connection_info/5.
%  - fails if connection can't be made
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_ListenAt(ConnectionId, tcp(Host,Port)) :-
      ground(ConnectionId),
      % if variable address, look it up...
      ((var(Host) ; var(Port)) ->
         com_ResolveVariables([
```

```
            [cmd('-oaa_host',Host), cmd('-oaa_port', Port)],
            [env('OAA_HOST', Host), env_int('OAA_PORT', Port)],
            [setup('setup.pl',oaa_host, Host),
             setup('setup.pl',oaa_port, Port)]
        ])
      | true),

      repeat,
      (on_exception(E,
                tcp_listen_at_port(Port, Host),
                Exception = E) ->
         ( var(Exception) ->
            assert(com_connection_info(ConnectionId, tcp, server,
               [addr(tcp(Host,Port)),oaa_host(Host),oaa_port(Port)],
               connected)),
            !
         | otherwise ->
            com_ask_about_tcp_exception(Port, Host, Response),
            ( Response == yes ->
                  fail
            | otherwise ->
                  halt
            )
         )
      |
            com_ask_about_tcp_exception(Port, Host, Response),
            ( Response == yes ->
                  fail
            | otherwise ->
                  halt
            )
      ).


com_ask_about_tcp_exception(Port, Host, Response) :-
      repeat,
      with_output_to_chars(
             format('Currently unable to access ~w port ~w.~n  Try again? ~w',
                  [Host, Port, '[y)es, n)o, h)elp]']),
            Chars),
      name(Prompt, Chars),
      ask_oneof(Prompt, [yes, no, help], Response),
      ( Response == help ->
            com_print_tcp_exception_help,
            fail
      | otherwise ->
            !
      ).

com_print_tcp_exception_help :-
    write('
I''ve just attempted to listen on the specified port, but was unable
to gain control of it.  This could be because there''s already a
Facilitator, or some other program, making use of that port.  Or, it
could be that a Facilitator using that port was just terminated.  In
such cases, the port may be inaccessible for a brief period (usually
only a few seconds, but sometimes more).  It may help to kill any
```

```
client agents which may still be connected to the defunct Facilitator.

If you think the specified port may now be accessible, enter "y" and
I''ll try again.  You may request retry any number of times.

If you want me to listen on a different port, enter "n", which will
cause me to terminate.  Then change your port specification (it''s
either in a setup file or an environment variable).  Then restart me.

').



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_SendData(+ConnectionId, +Data)
% purpose: Sends data to the specified connection ID
% remarks:
%  - Checks format for destination connection
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_SendData(ConnectionId, Data) :-
      ground(ConnectionId),
      ( com_connection_info(ConnectionId, Type, _ClientServer, InfoList,
           connected),
        (Type = tcp ; Type = unknown), !,
        memberchk(connection(Dest), InfoList)
        ;
        format('~nError: cannot find open connection for ~p!~n',
           [ConnectionId]),
        fail
      ),
      ( memberchk(format(F), InfoList) ->
          true
      | memberchk(agent_language(c), InfoList) ->
          F = special_case_c
      | otherwise ->
          F = default
      ),
      !,
      com_send_data_by_format(Dest, F, Data).


% quintus_binary: for inter-quintus communication
com_send_data_by_format(Dest, quintus_binary, Data) :- !,
      tcp_send(Dest, Data).
% prolog: a synonym for quintus_binary
com_send_data_by_format(Dest, prolog, Data) :- !,
      tcp_send(Dest, Data).


% pure_ascii: don't wrap data in term() wrapper
com_send_data_by_format(Dest, pure_ascii, Data) :- !,
        current_output(CurrentOutput),
        flush_output(CurrentOutput),
        tcp_output_stream(Dest, TcpOutput),
        set_output(TcpOutput),
```

```
        WriteParams =
           [quoted(true),            % make input acceptable for read
            ignore_ops(false),       % false so list will be printed as '[1,2]'
            % !!! could be a problem with +, other opts.
            numbervars(true),        % print vars as f(A).
            character_escapes(false),% write actual character, not \255
            max_depth(0)],           % no depth limit

        write_term(Data, WriteParams),

        flush_output(TcpOutput),
        set_output(CurrentOutput), !.

% special_case_c: This is the same as default, EXCEPT for the use of
% nl, nl.  See comments within the clause for default format.
% Currently we don't understand why it matters.
com_send_data_by_format(Dest, special_case_c, Data) :- !,
        current_output(CurrentOutput),
        flush_output(CurrentOutput),
        tcp_output_stream(Dest, TcpOutput),
        set_output(TcpOutput),

        WriteParams =
           [quoted(true),            % make input acceptable for read
            ignore_ops(false),       % false so list will be printed as '[1,2]'
            % !!! could be a problem with +, other opts.
            numbervars(true),        % print vars as f(A).
            character_escapes(false),% write actual character, not \255
            max_depth(0)],           % no depth limit

        write_term(term(Data), WriteParams),
        write('.'),
        nl, nl,
        flush_output(TcpOutput),
        set_output(CurrentOutput), !.

% DefaultOAA: wrap in term() wrapper for easy parsing
com_send_data_by_format(Dest, _DefaultOAA, Data) :-
        current_output(CurrentOutput),
        flush_output(CurrentOutput),
        tcp_output_stream(Dest, TcpOutput),
        set_output(TcpOutput),

        WriteParams =
           [quoted(true),            % make input acceptable for read
            ignore_ops(false),       % false so list will be printed as '[1,2]'
            % !!! could be a problem with +, other opts.
            numbervars(true),        % print vars as f(A).
            character_escapes(false),% write actual character, not \255
            max_depth(0)],           % no depth limit

        write_term(term(Data), WriteParams),
        write('.'),
        % nl, nl,
% The preceding does not work between two Quintus agents
% (neither does a single nl, nor does it help to use nl(TcpOutput)),
% so we went to the following.  However, the following does not work
```

```
% when a QP facilitator sends to the C interface agent.  For now,
% we'll solve this problem by defining the special_case_c format.
% (DLM, 97-04-09)
      put(TcpOutput, 10),
      % This causes the agents to disconnect (at least under UNIX):
        % put(TcpOutput, 13),

        flush_output(TcpOutput),
        set_output(CurrentOutput), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_SelectEvent(+TimeOut, -Event)
% purpose: Waits and returns an incoming event, or 'timeout' if TimeOut expires
% remarks:
%  - TimeOut may be a real number, and represents seconds.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_SelectEvent(0, Event) :- !,
      on_exception(E,tcp_select(Event), com_print_err(E)).
com_SelectEvent(Seconds, Event) :-
      on_exception(E,tcp_select(Seconds, Event),com_print_err(E)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_print_err
% purpose:  Print error message if problem reading the event
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_print_err(E) :-
   format('~n=========== READ ERROR !!! ============~n',[]),
   format('| Messages in this block are rejected~n',[]),
   format('| by the system.~n',[]),
   format('----------------------------------------~n',[]),
   print_message(error, E),
   format('========================================~n',[]), fail.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_AddInfo
% purpose:  Adds or changes information about connection
% remarks:
%    Info may be status(S), type(T), protocol(P) or any element (or list
%    of elements) to be stored in InfoList.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_AddInfo(ConnectionId, NewInfo) :-
      retract(com_connection_info(ConnectionId, Protocol, Type,
            InfoList, Status)),
      (NewInfo = status(NewStatus), C = true ; NewStatus = Status),
      (NewInfo = protocol(NewProtocol), C = true ; NewProtocol = Protocol),
      (NewInfo = type(NewType), C = true ; NewType = Type),
      (NewInfo = [_H|_T] ->
         union([InfoList, NewInfo], NewInfoList)
       | (ground(C) ; union([InfoList, [NewInfo]], NewInfoList))
      ),
      assert(com_connection_info(ConnectionId, NewProtocol, NewType,
```

```
            NewInfoList, NewStatus)), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_GetInfo
% purpose:  Looks up information about connection
% remarks:
%    Info may be status(S), type(T), protocol(P) or any element stored
%    in InfoList.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_GetInfo(ConnectionId, Info) :-
      com_connection_info(ConnectionId, Protocol, Type,
            InfoList, Status),
      (Info = status(Status) ;
       Info = type(Type) ;
       Info = protocol(Protocol) ;
       memberchk(Info, InfoList)),
      !.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% name:    com_ResolveVariables
% purpose: Tries to instantiate the arguments by looking in the command
%        line arguments, environment variables, and setup files
%  inputs:
%   - VarList:    A list of lists: the first sublist that completely resolves
%        provides the value for com_ResolveVariables.
% remarks:
%    sublists may contain elements in the following format:
%       env(EnvVar, Val)       : looks for "EnvVar" in environment vars
%     env_int(EnvVar, Val)     : Returns value for EnvVar as an integer
%     cmd(CmdVar, Val)   : looks for "CmdVar <Val>" on command line
%     setup(File,SVar, Val)    : reads SVar from setup file File
% example:
%    resolves host and port by searching first commandline, then environment
%    variables, finally reads setup file.
%
%     com_ResolveVariables([
%         [cmd('-oaa_host',Host), cmd('-oaa_port', Port)],
%       [env('OAA_HOST', Host), env_int('OAA_PORT', Port)],
%       [setup('setup.pl',oaa_host, Host),
%        setup('setup.pl',oaa_port, Port)]
%     ])
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_ResolveVariables([VarList|_]) :-
      com_resolve_variables(VarList), !.
com_ResolveVariables([_VarList|Rest]) :-
      com_ResolveVariables(Rest).

com_resolve_variables([]).

com_resolve_variables([env_int(EnvVar, Val)|Rest]) :- !,
      environ(EnvVar, EnvAtom),
      name(EnvAtom, EnvChars),
```

```
      number_chars(Val, EnvChars),
      com_resolve_variables(Rest).

com_resolve_variables([env(EnvVar, Val)|Rest]) :- !,
      environ(EnvVar, Val),
      com_resolve_variables(Rest).

com_resolve_variables([cmd(CmdVar, Val)|Rest]) :- !,
      % get command line arguments
      unix(argv(ListOfArgs)),
      append(_, [CmdVar, Val|_], ListOfArgs),
      com_resolve_variables(Rest).

com_resolve_variables([setup(File,SVar, Val)|Rest]) :- !,
      % read setup file to load all values
      com_read_setup_file(File),
      Pred =.. [SVar, Val],
        on_exception(_, Pred, fail),
      com_resolve_variables(Rest).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    com_read_setup_file
% purpose: Finds and loads setup file
% remarks:
%    Always succeeds.
%    The search path for 'setup.pl' is as follows:
%      1. Current directory
%      2. Home directory for user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_read_setup_file(File) :-
      com_already_loaded(File), !.
com_read_setup_file(File) :-
      ( absolute_file_name(File, LocalSetupFile),
        can_open_file(LocalSetupFile, read, fail) ->
          SetupFile = LocalSetupFile
      |
        concat('~/',File, HomeName),
        absolute_file_name(HomeName, UserSetupFile),
          can_open_file(UserSetupFile, read, fail) ->
             SetupFile = UserSetupFile
      ),

      (ground(SetupFile) ->
         format('Loading setup file:~n  ~w~n~n', [SetupFile]),
         ( com_consult(SetupFile, _) ->
             assert(com_already_loaded(File))
         | otherwise ->
             format('~w: A problem was encountered in loading the setup file~n',
                  ['WARNING'])
         )
      | true).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% name:    com_consult(+FilePath, -AbsFileName).
% purpose:
% remarks: We don't use Quintus' builtin consult, because it's too picky
%          about associating predicates with files.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
com_consult(FilePath, AbsFileName) :-
    absolute_file_name(FilePath, AbsFileName),
    can_open_file(AbsFileName, read, fail),
    open(AbsFileName, read, Stream),
    load_clauses(Stream),
    close(Stream).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    load_clauses(+Stream).
% purpose:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_clauses(Stream) :-
    repeat,
    read_term(Stream, [], Term),
    ( Term = ':-'(_Body) ->
      true
    | Term = end_of_file ->
      true
    | otherwise ->
        load_clause(Term)
    ),
    ( at_end_of_file(Stream) ->
      !
    | otherwise ->
      fail
    ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    load_clause(+Term).
% purpose:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_clause(Term) :-
    assert( Term ).
```